Open-Apple™

May 1987
Vol. 3, No. 4

ISSN 0885-4017
newstand price: $2.00
photocopy charge per page: $0.15

Releasing the power to everyone.

Verses added to a golden oldie

At the very core of every Apple II ever built, from the first Integer Basic model to the IIgs, you'll find something called the System Monitor. This built-in software allows you to examine and control the Apple II at a very intimate level. Citizens of other computer kingdoms, who don't have similar programs built into their computers, will tell you the Apple Monitor has little significance—they'll say only assembly language high priests can use it. In fact, just the opposite is true.

The significant thing about the Apple Monitor is that it has lured tens of thousands of mere mortals into learning how the Apple II works. There are no secrets in our kingdom (it's a tradition handed down from Woz) and there are no high priests. The Monitor is one of the major tools used by laypeople to learn how to release the power of the Apple II. Likewise, it is this body of thousands of laypeople who have learned how to release the Apple II's power that make the Apple II different from the machines at the center of other kingdoms.

Being one of the mere mortals myself (computer high priests rarely accept English majors as one of their own), I've always had a great deal of respect for the Monitor. Way back at the beginning of ***Open-Apple's*** Volume 1 you'll find two articles, "A Song Called the System Monitor" (February 1985, pages 1.9-1.12) and "A Song Continued" (March 1985, pages 1.20-1.21), that were written to introduce the Monitor to those of you who had never used it. The articles describe the Monitor as it exists on the original Apple II, the II-Plus, the IIe, the IIc, and the enhanced IIe.

This month we're going to summarize what was explained in detail in those issues and we're going to take a closer look at the Monitor as it exists on the new Apple IIgs. The Monitor changed hardly at all between the original Apple II and the enhanced IIe, even though the machines themselves progressed through several revisions. For example, the memory capacity of the machines grew from 48K to 64K to 128K, but the memory-examining capacity of the Monitor never grew beyond the original 48K.

Apple has fixed all this in the IIgs, however. The IIgs Monitor has been enhanced with new powers and new commands. On the IIgs you can examine and manipulate memory in the "language cards" and in the "auxiliary bank" as easily as any other memory. In addition, you'll find memory displays that can take advantage of 80-column screens and show ASCII values, commands for converting numbers back and forth between hex and decimal, and several commands that support new features of the IIgs.

Entering the Monitor. The usual way of entering the Monitor is from Applesoft with a CALL -151 command. The entry point of the machine language program that is the Monitor starts at $FF69, which is equivalent to -151 in decimal (or 65385, take your pick).

Another way to get into the Monitor is to get the microprocessor to execute a BRK (break) command. The microprocessor will do this whenever it encounters a zero (the machine code for BRK) as it is executing a machine language program. Most users think of this as "crashing into the Monitor." When the program you are using beeps, puts an asterisk on the screen, and displays a new row of letters and numbers every time you press return, this is what has happened to you. Unless you have a IIgs, you'll see a line that looks something like this somewhere on your screen:

```
0802-    A=0B X=15 Y=25 P=30 S=E4
```

On a IIgs, the line looks more like this:

```
00/0800: 00 00        BRK 00
 A=0008 X=0015 Y=0025 S=01E4 D=0000 P=30 B=00 K=00 M=0C Q=80 L=1 m=1 x=1 e=1
```

Machine language programmers use BRK for debugging purposes. A BRK stops a program at a spot selected by the programmer and allows examination of the microprocessor's memory registers and the computer's memory. When a commercial program executes a BRK and crashes into the Monitor, it's a bug. Finished programs aren't supposed to do that.

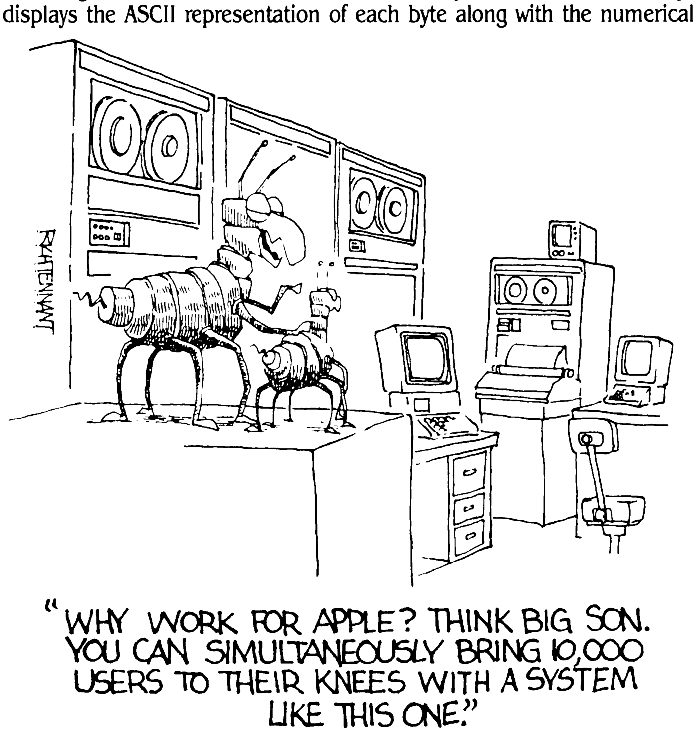
A third way to get into the Monitor, but only on the IIgs, is with a desk accessory called *Diversi-Hack*. Interrupt **any** program with open-apple/control/escape to get to the IIgs desk accessory menu, choose *Diversi-Hack* from the menu, and you are in the Monitor. The IIgs Monitor's new Q(uit) command will take you back to the desk accessory menu; a quit from there will take you back to the program in progress. For more about *Diversi-Hack*, see my introduction to this month's letters.

Once you get into the Monitor, you'll find that its commands fall into four groups. There are commands for examining memory, commands for changing memory, commands for program control, and miscellaneous commands.

Examining Memory. The values found in a computer's memory cells can represent just about anything, but most often they represent either numbers, ASCII characters, or machine language programs. The IIgs Monitor gives you tools to examine RAM and ROM memory from each of these three perspectives. (The Monitor on older Apples doesn't include an ASCII display, but ***Open-Apple*** already solved that problem back on page 1.12. If you don't know RAM from ROM from registers, go back to the beginning of our second volume and try "The Magic of Peek and Poke," February 1986, pages 2.2-2.5.)

To look at memory from the perspective of numbers, you enter the hex address that you want to see the contents of and press return. Successive returns display successive bytes of memory. To display a range of memory all at once, enter the beginning and ending addresses separated by a period.

The IIgs offers several enhancements to this system. First of all, the IIgs displays the ASCII representation of each byte along with the numerical



"WHY WORK FOR APPLE? THINK BIG SON. YOU CAN SIMULTANEOUSLY BRING 10,000 USERS TO THEIR KNEES WITH A SYSTEM LIKE THIS ONE."

value. If you are in 80-column mode, the Monitor will notice it and will display 16 bytes on each line instead of the usual eight. Finally, pressing control-X will terminate the display of a range of memory values. On older Apples, the only way to terminate a range display is to press control-reset. Here is what the 80-column memory display looks like on the IIgs:

```
*2000.20FF
00/2000:00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F-................
00/2010:10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F-................
00/2020:20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F- !"#$%&'()*+,-./
00/2030:30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F-0123456789:;<=>?
00/2040:40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F-@ABCDEFGHIJKLMNO
00/2050:50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F-PQRSTUVWXYZ[\]^_
00/2060:60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F-`abcdefghijklmno
00/2070:70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F-pqrstuvwxyz{|}~.
00/2080:80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F-................
00/2090:90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F-................
00/20A0:A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF- !"#$%&'()*+,-./
00/20B0:B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF-0123456789:;<=>?
00/20C0:C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF-@ABCDEFGHIJKLMNO
00/20D0:D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF-PQRSTUVWXYZ[\]^_
00/20E0:E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF-`abcdefghijklmno
00/20F0:F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF-pqrstuvwxyz{|}~.
```

The ASCII characters on the right side of the display represent the same sixteen characters that appear numerically in the midsection of the display. This example shows a range of memory that I purposefully filled with values from 0 to 255 so you could see the relationship between values and ASCII characters. Control-characters are displayed as periods. No distinction is made between high-value and low-value ASCII.

The numbers on the left edge represent the memory address of the first of the sixteen bytes in each line. The 6502 microprocessor used on earlier Apples could accommodate only 65,536 discreet memory cells. Thus Monitor addresses on these Apples ran from $0000 to $FFFF. The microprocessor in the IIgs, on the other hand, can accommodate 16,777,215 discreet memory cells. These are arranged as 256 "banks" of 65,536 cells each. The number in front of the slash indicates which bank you are looking at. The number after the slash gives the address within that bank.

To look at a different bank, you can enter a number such as 02/2000.20FF. This would display "page" $20 in bank $02. The addresses you give must both be within the same bank.

L(ist). The third perspective for looking at memory is with the L(ist) command. Enter an address and "L" and you will see a "disassembly" of the section of memory that follows that address. Not all ranges of memory actually hold machine language programs. Some hold data, some hold programs in other languages, some hold nothing at all. The big limitation of the list command (a limitation it shares with other disassemblers) is that it can't tell real machine language code from other kinds of data. It "disassembles" everything, whether the results are meaningful or not.

The IIgs list command is able to disassemble all the 65816 operation codes and addressing modes. For a complete description of these codes and modes see "A 65802/65816 pre-boot" and "Introduction to the 65802/65816" in our August 1986 issue, pages 2.49-56.

One big problem that occurs when a program tries to disassemble 65816 machine code is that the "immediate mode" of three commonly-used instructions, LDA, LDX, and LDY, can be either two or three bytes long and the disassembler can't tell which. These commands load the microprocessor's A, X, or Y register. With immediate addressing, the data that is to be put into the register is embedded within the program immediately after the instruction byte. If the registers are set up for 8-bit data, these instructions are two bytes long (one instruction or operation-code byte and one data byte). If the registers are set up for 16-bit data, on the other hand, these instructions are three bytes long (two data bytes). But a disassembler can't tell which is which. For example, get into the IIgs Monitor and try this:

```
*FF/0203L
1=m   1=x   1=LCBANK (0/1)

FF/0203: A9 00         LDA #00
FF/0205: 00 A2         BRK A2
FF/0207: 00 05         BRK 05
```

Those BRK instructions in the second and third lines don't look quite right. Perhaps this section of code was written to be executed with 16-bit registers. To tell the disassembler that you'd like to see the code listed that way, enter 0=m (16-bit A register) and 0=x (same for X and Y registers). Note that the format of these commands is backwards from what Applesoft programmers would expect (0=m, not m=0), that "m" and "x" *must* be entered in lower case, and that the current list status of "m" and "x" is always displayed at the top of a listing. For example:

```
*0=m 0=x 0203L
0=m   0=x   1=LCBANK (0/1)

FF/0203: A9 00 00      LDA #0000
FF/0206: A2 00 00      LDX #0500
```

When you examine memory with the Monitor and you look at bank zero, you will *always* see the "language card" RAM in the address space from $D000 to $FFFF. If what you really want to see is the ROM that is normally found in this address space, look for it in bank $FF. That's what we've done here. (There's more ROM in bank $FE.) Note in the above examples that once you've set the bank to $FF (or whatever), it will stay there (with one exception to be noted later) until you change it again.

Unlike the ROM code, the hardware softswitches and peripheral card firmware that appear in the address range from $C000 to $CFFF on earlier Apples appear there still on the IIgs (but in banks $00, $01, $E0 and $E1 only). Thus, two banks of $D000-$DFFF memory are needed to squeeze 64K of RAM into these banks. To control which bank you are looking at, enter 1=L to see the main bank (usually called bank 2 in the Apple literature) and 0=L to see the secondary bank (usually called bank 1).

In addition to allowing you to examine sequential memory bytes as values, ASCII characters, and assembly language mnemonics, the IIgs Monitor gives you the power to examine memory by searching for a specific byte pattern and by comparing two segments of memory for differences.

P(attern search). The first Apple II Monitor to have a search command was the one in the enhanced IIe. On that machine you can search memory for any one-byte or two-byte value. The command's syntax is "YYXX<adr.adrS" where YYXX is two sequential bytes appearing as XX YY. Yes, they're backwards.

The IIgs search command is very different from the one in the enhanced IIe. First of all, rather than being known as S(earch), as on the enhanced IIe, it goes by P(attern search). Second, the values (or pattern) you are searching for can be up to 236 bytes long. Third, the pattern you are looking for must be surrounded by backslashes. Fourth, the pattern can include hexadecimal values, ASCII values, or "flipped" ASCII (flipped means backwards — "BOB" instead of "BOB," for example). Try this:

```
*\20 ED FD\<FF/F800.FFFFp  search Monitor code for JSR $FDED

FF/F94C:
FF/FD4A:
FF/FD64:
FF/FD6C:
FF/FF2F:
FF/FF34:
FF/FF37:

*\"Apple"\<FF/0000.FFFFp   search bank $FF for "Apple"

FF/8B02:
FF/89A7:
FF/C767:
FF/F914:
FF/FB09:
```

When you search for ASCII characters as we did above, the search finds only high-value ASCII characters. However, the IIgs Monitor has an "ASCII filter mask" that will modify the characters you specify to low-value ASCII or to other weird stuff. The filter mask goes by the name "F." It can be set to anything from $00 to $FF with the =F command. $FF is its natural setting. $7F will get you low-value ASCII. The other 254 settings are of questionable value. Try this:

```
*7F=F
*\"Apple"\<FF/0000.FFFFp   search bank $FF for "Apple"

FF/2A19:
FF/9128:

*\'PDGO'\<B000.BFFFp        search 00/B000 to BFFF for "OGOP"

00/B893:
```

Notice a couple of things here. The search for "Apple" with the filter set to $7F turned up two more occurrences that the first search didn't catch. Likewise, none of the first search's hits turned up in the second search. This means the "ASCII filter mask" doesn't create "wildcard" bits. Instead, it

simply clears bits in the character pattern specified. Only exact matches with the new pattern will be found. Secondly, notice that in the search for POGO backwards (the backwards part is specified by means of a single quote mark rather than a double quote mark) we didn't specify a bank address, but in all the other searches we did. This is because the IIgs search command always defaults to bank zero unless you specify another bank. This is inconsistent with the other Monitor commands, which default to the last bank specified, and is the exception to the last-bank-specified default that was mentioned earlier. This is probably a bug, but maybe there's a good reason for it I haven't discovered yet.

V(erify). The command for comparing, or "verifying" two segments of memory hasn't changed from earlier Apple IIs, except that bank addresses can now be specified. If "adr" means an address of either the form "XXXX" or "XX/XXXX," then the format of this command is "adr<adr.adrV". The two segments of memory you are comparing can be in different banks, but neither segment can cross a bank.

For example, load an image of Applesoft as found on the IIc into a IIgs at $2000. (More on how to actually do this later.) To compare this image to what's in the IIgs, do this:

```
*00/2000<FF/D000.F7FFv
FF/E006:00 (96)
FF/F1FE:30 (50)
FF/F233:D9 (09)
FF/F234:F8 (F2)
FF/F3CB:20 (AD)
FF/F3CC:3A (79)
FF/F3CD:F9 (C0)
```

As you can see from the display, the two images are almost exactly the same. For a complete description of the differences between IIc Applesoft and earlier versions, see "Up where Applesoft lives" in our June 1985 issue, pages 1.40-44. The primary difference is that IIc Applesoft (and now Applesoft on the IIgs) includes support for double-low-resolution graphics (80 pixels across the screen by 48 high).

(The differences between IIc Applesoft and IIgs Applesoft are as follows: $E006 is a spare byte that has had various values in various versions of Applesoft—the zero in this byte on the IIgs matches what the original Applesoft had. The next three changes fix a bug in the double-low mode that allowed the Y coordinate to be as large as 80 on PLOT and SCRN commands, when 48 should have been the maximum. The last three changes were necessary because the softswitch for reading whether double-resolution graphics are turned on isn't in the same place on the IIc and IIgs. Mysteriously, a one-byte change necessary to fix a bug in the double-low SCRN function, which was mentioned in our June 1985 article, wasn't done.)

Modifying Memory. There are several ways you can use the Monitor in all models of Apple II to modify RAM memory. To change one byte, enter the address of the byte you want to change, a colon, and the new value. To change the bytes that follow the first, enter more values, but be sure to include a space between each. After you press return you can continue where you left off by simply entering a colon and more values.

In addition to entering hexadecimal numbers, the IIc, enhanced IIe, and IIgs have what's called an "ASCII input mode." On the IIc and enhanced IIe you can store the ASCII code for a letter in memory by typing the letter with a single quote in front of it. If you want to enter several letters in a row, you must precede each with a single quote, like this:

```
*2000:'D 'r 'a 'g 'o 'n '' 's      IIc/enhanced IIe ASCII input mode
*2000.2007
2000- C4 F2 E1 E7 EF EE A7 F3
```

ASCII input mode on the IIgs is much different. Instead of preceding each letter with a single quote, you surround the letters with double quotes. The characters you enter are marched through the ASCII filter mask (mentioned earlier in the discussion of the search command) before being stored in memory—thus you can use the filter to obtain low-value ASCII. Only high-value ASCII is available on the IIc and enhanced IIe. Here are some examples:

```
*07/100:"Dragon's"
*100.10F
07/0100:C4 F2 E1 E7 EF EE A7 F3 FF FF FF FF FF FF FF FF-Dragon's........

*110:'Dragon"s'
*110.11F
07/0110:F3 A2 EE EF FF FF FF FF FF FF FF FF FF FF FF FF-s"no............

7F=F
*120:"Dragon's
*120.12F
07/0120:44 72 61 67 6F 6E 27 73 FF FF FF FF FF FF FF FF-Dragon's........
```

As you can see in the second example, a single quote is supposed to give you backward, or flipped, ASCII. However, it seems to work only with the last four characters you enter. That's a bug. In the third example, I changed the filter mask to get low-value ASCII. Notice that you don't really need a closing quotation mark if you're at the end of a line. (In comparing the IIgs with the IIe and IIc I just noticed another important Monitor difference—on the IIgs the keyboard's delete key works, at last, at last.)

M(ove). The move command, which is available on all versions of the Monitor, provides another way to modify memory. The command syntax is "dest<adr.adrM" where "dest" is the destination address and the "adr.adr" range defines the segment of memory you want to move. On the IIgs this command can move memory from bank to bank, but just as with the verify command, it can't deal with ranges that cross a bank boundary. It's also important to make sure the destination area isn't inside the source area, or you may get results other than those you expected.

Earlier I mentioned that I'd show you how to get an image of IIc Applesoft into the IIgs, so that the two can be compared. You can't just BSAVE IIC.FP.IMAGE,A$D000,L$2800 because while the BSAVE is going on ProDOS will be active. The snapshot you'll get will be of the ProDOS kernal rather than of Applesoft. This can be bewildering. The solution is to either take the snapshot with DOS 3.3, or do this:

```
*2000<D000.F7FFm      move Applesoft's image to $2000 (do this on a IIc)
*3D0G
]BSAVE IIC.FP.IMAGE,A$2000,L$2800
```

Z(ap). The IIgs has a new command for filling a range of memory with a specific value. The syntax is "val<adr.adrZ", where val is a hex number. For example, 7<07/0000.FFFFz will put a seven in every byte in bank seven, which is not necessarily a good thing to do. You can also zap memory on older Monitors, but it requires a trick. Try:

```
*2000:0
*2001<2000.20FEM
```

Because the destination address is inside the source address, the effect of this command will be to move the value at byte $2000 into every byte on that page.

The Mini-Assembler. The final way to modify memory with the Monitor is to use the Mini-Assembler. The heritage of the Mini-Assembler goes back to the original Apple II, where it could be found as part of Integer Basic. You can run the Integer Basic Mini-Assembler by entering the Monitor while Integer Basic is active and typing "F666G". Because it uses "undocumented entry points," however, the Integer Basic Mini-Assembler doesn't work right on the IIc or IIgs. On the enhanced Apple IIe, the 3.5 ROM IIc, and the IIgs, you can start up newer versions of the Mini-Assembler directly from the Monitor, with or without Integer Basic, by entering an exclamation point.

The Mini-Assembler can be used to write short, uncomplicated assembly language programs. It's much easier to write longer programs on full-blown assemblers because they allow you to give names or "labels" to program segments and addresses, because they make it easy to insert new lines in what you've already written, and because they provide many other amenities that the Mini-Assembler doesn't. All the Mini-Assembler can do is allow you to type in one assembly language mnemonic and operand at a time; it converts these into the equivalent machine language values and stores them in memory. Nonetheless, it's a useful and handy tool to have around.

It was possible to execute Monitor commands from within the original Mini-Assembler by starting a line with a dollar sign. This isn't possible with the Mini-Assemblers that start up with an exclamation point. However, it's not necessary either because you can switch between the Mini-Assembler and the Monitor so easily on these machines (the exclamation point gets you in, return on a blank line gets you out). In addition, the IIgs Mini-Assembler does allow you to directly enter hex or ASCII values into memory.

Once you are in the Mini-Assembler, the first thing you must do is provide the address where you want your machine code to be placed. Follow that with a colon and an assembly language instruction. The allowable instructions, addressing modes, and addressing mode formats for the Mini-Assembler are shown in our August 1986 issue, page 2.52. The IIgs Mini-Assembler can handle all of the 65816 instructions and addressing modes; the 3.5 IIc model can handle all 65C02 instructions; other models handle only the 6502 instructions and modes.

Begin each line you enter after the first with a blank space if you want that line's machine code to follow the previous line's. On the IIgs, a colon instead of a blank space tells the Mini-Assembler that hex data follows. A double quote mark instead of a blank indicates ASCII characters (which will be forced through the ASCII filter mentioned earlier). A number indicates you are giving a new address for storing machine code. Thus:

```
*!
!2000:LDX #0
00/2000: A2 00        LDX #00          load a zero into X register

! LDA 200E,X
00/2002: BD 0E 20     LDA 200E,X       load A with what's at $200E+X

! BEQ 200D
00/2005: F0 06        BEQ 200D {+06}   branch on equal--did A get a zero?

! JSR FDED
00/2007: 20 ED FD     JSR FDED         jump to the PRINT subroutine at $FDED

! INX
00/200A: E8           INX              increment X (X=X+1)

! BNE 2002
00/200B: D0 F5        BNE 2002 {-0B}   if X<>0, branch to get next character

! RTS
00/200D: 60           RTS              return to caller

!"Read Open-Apple every month for health and wealth.
!:0
```

How did I know while I was entering the second line that the ASCII string the routine prints would occur at $200E? I didn't. This is the kind of thing a true assembler handles easily. In this case, it's necessary to guess at the correct address (in order to save space for the instruction), then go back and correct it after you have finished the program and know the correct address. It's also necessary to guess at and correct the destination of the branch in the third line of the program. Incidentally, the Mini-Assembler display you see on your screen is much cleaner than what's possible to show here—the output for each line overwrites the input. Try it and see.

Program control. Once you've entered the above program with the Mini-Assembler, press return on a blank line to get back inside the Monitor and try this:

```
*2000g
Read Open-Apple every month for health and wealth.
```

The G(o) command tells the Monitor to execute the subroutine at the address you give. $2000 is the temporary home of the Chinese fortune subroutine we just entered with the Mini-Assembler.

An important, but often overlooked, aspect of the (G)o command is that it loads the microprocessor's registers with specific values, which you can control, just before jumping to the subroutine at the address you specify. (Incidentally, it *does* JSR, not JMP, so if the routine you call ends with an RTS you'll return cleanly to the Monitor, as in the example here.)

Try this:

```
*(press control-E and return)

---display on II, II-Plus, IIe, IIc, enhanced IIe
 A=08 X=15 Y=25 P=30 S=F0

---display on 3.5 IIc
 M=00 A=08 X=15 Y=25 P=30 S=F0

---display on IIgs
 A=0008 X=0015 Y=0025 S=01F0 D=0000 P=30 B=00 K=00 M=0C Q=80 L=1 m=1 x=1 e=1
```

Control-E displays the values that will be placed in the microprocessor's registers when control is passed to the address you specify with the G(o) command. On the 3.5 IIc and the IIgs some of the displayed values aren't actually registers, but are "flags" that indicate which memory banks will be active when G(o) is executed, as well as other stuff. You see this same display after the microprocessor hits a BRK instruction, as mentioned many paragraphs ago. Not only is the display the same, so are the values. In other words, a BRK puts you into the Monitor and displays the values in the registers when the BRK occurred. The flags indicate the status of the machine at that time. G(o) returns to the program with those same values in the registers and that same machine status.

You can, however, *change* the values from the Monitor if you like, so that G(o) will use different values or flags. On all machines except the IIgs you do this by pressing control-E and return to get the register display. Then enter a colon at the beginning of the next line followed by the value you want in the A register, the value you want in the X register, and so on. If you'd like to experiment with this, be aware that changing the P or S register to a random number sometimes locks the machine up so tight you need a can opener to get it back open. (To change the M register on the 3.5 IIc use "44:val".)

On the IIgs, on the other hand, you change the values that will be placed in the registers by entering the new value you want, an equal sign, and a letter designating the register you want to change. We used this format earlier in our discussions of the L(ist) command and the ASCII filter. On the IIgs you can also restore the registers and flags to a "normal" configuration by pressing control-R.

The program we entered earlier with the Mini-Assembler begins by loading the X register with a zero. You can see what would happen with other values by changing the X register value with the Monitor and G(o)ing to $2002. For example:

```
---all but IIgs                  ---IIgs
*(control-E return)              *(control-E return is optional on IIgs)
(registers are displayed)        (registers are displayed)
*:00 05                          *5=X
*2002G                           *2002G

Open-Apple every month for health and wealth.
```

The A register is the microprocessor's accumulator, where all math operations are done. X and Y are the index registers. S is the stack register, which points a crooked finger at the current stack position. P is the microprocessor's status register. The meaning of its bits change slightly, depending on whether the microprocessor is in 6502 ("emulation") mode or 65816 ("native") mode. Only the IIgs has these two modes. The "e" flag tells you which mode the machine is in; 1=6502 mode, 0=65816 mode. Here's what the bits in the P register mean:

```
Meaning of the P(rocessor status) register

N V 1 B D I Z C   e=1 on IIgs (6502 mode) and all other Apple IIs
N V M X D I Z C   e=0 on IIgs (65816 mode) only

N is sign; 1=negative
  V is overflow; 1=true
    1 is unused
    M is A-register width; 1=8 bits, 0=16 bits
      B is break flag; 1=BRK, 0=hardware interrupt
      X is X-register width; 1=8 bits, 0=16 bits
        D is binary coded decimal flag; 1=true
          I is interrupt flag; 1=interrupts disabled
            Z is zero flag; 1=true
              C is carry flag; 1=true
```

The bits of the M register indicate the state of the machine's memory banks when a BRK occurred, or how you want the banks arranged for the next G(o) command. Both the 3.5 IIc and the IIgs display an M register. Here's what the bits mean.

```
Meaning of the M(emory status) register

00 P2 RD WR LC B1 B2 00   3.5 IIc only
AZ PX RD WR LX BX AR CX   IIgs only

00 is unused
AZ is alternate language card/zero-page/stack; 1=active
   P2 is STORE80/PAGE2 status; 1=both active
   PX is PAGE2 status only; 1=active
      RD is auxmem read status; 1=active
         WR is auxmem write status; 1=active
            LC is language card read status; 1=card active
            LX is language card status; 1=ROM active
               B1 is language card bank 1 read status; 1=active
               BX is overridden by L (see text); 1=bank 2, 0=bank 1
                  B2 is language card bank 2 read status; 1=active
                  AR is alternate ROM bank (see text); 1=active
                     00 is unused
                     CX is alternate $C100-$CFFF ROM; 1=active
```

The meaning of the bits in the 3.5 IIc's M register is similar to, but not exactly the same as, the M register on the IIgs. In particular, the LC/LX bits, which indicate whether ROM or RAM is active in the $D000-$FFFF memory area, have exactly opposite meanings. On the IIc a one in that bit means RAM is active, on the IIgs a one means ROM is active. The IIc uses two bits to indicate which language card bank is being used; if neither (if ROM is active), both bits are cleared to zero. The IIgs uses just one bit for this (1=bank 2, 0=bank 1), but changing that bit in the M register is a useless exercise—it's

always overridden by the L flag, which we looked at earlier in our discussion of the L(ist) command.

Both the 3.5 IIc and the IIgs have an alternate 16K ROM bank that can be turned on with softswitches. Only the IIgs has a bit in the M register to activate that ROM, however. On the IIgs a program can obtain all of the information in the Monitor's M register, in exactly the same format, by reading byte $C068. Not only that, but by writing to the same byte (which is called the "state" register), a IIgs program can change the machine's memory configuration. This provides Apple's programmers with a speedy way to save and restore the memory configuration of the IIgs during an interrupt.

The control-R(estore registers) command in the IIgs Monitor sets the M register to $08. A G(o) with M=08 would turn on the lower 48K of the main 64K bank of RAM and put the Applesoft/Monitor ROM in the $D000-$FFFF area.

The Q register (for "quagmire," according to the IIgs documentation) that appears in the IIgs memory display combines information from two other IIgs hardware registers. These are the "shadow" register at $C035 and the "configuration" register at $C036.

On the IIgs, what you see on your screen is always a reflection of information stored in RAM memory banks $E0 and $E1. (On other Apples the active video area is in banks $00 and $01.) Since Apple II programs written before the IIgs appeared don't know bank $E1 from Capitol Federal Savings and Loan, the IIgs hardware automatically "shadows" anything that is written into certain parts of banks $00 and $01 into the same parts of banks $E0 and $E1. It's as if you were able to deposit a dollar into First National and have your sugar daddy make a matching deposit into E-First National for you.

There may be situations, however, when you wouldn't want a matching deposit made. For example, if your program is using the high-resolution graphics page 1 memory area ($2000-$3FFF) for data instead of pictures, there may be no reason to have the data shadowed into E0/2000-3FFF as well. With shadowing on, anything previously stored in the $2000-3FFF area of bank $E0 would be destroyed by writes to 00/2000-3FFF. Shadowing also slows down the IIgs slightly. Under ProDOS 16, shadowing is normally turned off and video display manipulations are made directly to banks $E0 and $E1. Under ProDOS 8, DOS 3.3, and Pascal, shadowing is normally turned on.

Most of the bits in Q come from the shadow register. Only one, the one that indicates machine speed, comes from the configuration register. Here's the meaning of the bits in the quagmire register:

```
Meaning of the Q(uagmire) register

SP LM 00 AX SH H2 H1 T1

SP is processor speed; 1=high, 0=normal
   LM is linear memory; 1=no I/O space at $C000 in banks $00 and $01
      00 is unused, must be zero
         AX is auxmem hi-res override; 1=no hi-res auxmem shadowing
            SH is super hi-res (E1/2000-9FFF); 1=no shadowing
               H2 is main/aux hi-res page 2; 1=no shadowing
                  H1 is main/aux hi-res page 1; 1=no shadowing
                     T1 is main/aux text page 1; 1=no shadowing
```

Control-R sets the Q register to either $00 or $80. The speed is left as it was before the control-R.

Notice that it is possible, by manipulating the LM bit on the IIgs, to disable the language cards and the $C000-CFFF hardware in banks $00 and $01. If you did this, the RAM in banks $00 and $01 would become continuous—the primary language card bank (bank 2) would appear in the $C000 area and the secondary bank would be at $D000. The $C000 hardware would appear only in banks $E0 and $E1. However, this isn't a practical alternative because IIgs interrupts use some machine language code that lives in the $C000-CFFF ROM in bank zero. Interrupts cease to work when you invoke the linear memory option.

The other three registers shown in the IIgs register display are the direct register (D), the data bank register (B), and the program bank register (K). For more information on these registers, which are active only when the microprocessor operates in 65816 mode, see the August 1986 ***Open-Apple.***

X(ecute), R(esume), S(tep), and T(race). On the IIgs, the G(o) command can only be used to execute a routine in bank $00. If the routine you want to start lies elsewhere, use the X(ecute) command. This command expects the routine to end with an RTL (return from subroutine, long), however. Like X(ecute), R(esume) will also start up code in any bank. However, it JMLs (jumps long) rather than JSLing (jump to subroutine, long). Use it to continue program execution after a BRK. G(o) and X(ecute) don't work well after a BRK—they mess up the stack with their own JSR/JSL.

S(tep) and T(race) are available only in the original Apple II Monitor and in the 3.5 IIc Monitor. (Hooks were left in the IIgs for implementing these commands—they print "Step" and "Trace" on your screen at the moment.) S(tep) lets you execute machine language programs one instruction at a time. As each instruction is executed, it and the contents of the registers are displayed on the screen. 2000S, for example, would begin stepping through a program living at byte $2000. To execute the next instruction, simply press S and return.

T(race) is similar to S(tep) except that it doesn't stop after each instruction. To exit T(race) on the 3.5 IIc, press solid-apple. To slow it down to one step per second, press and hold down on open-apple. Neither S(tep) nor T(race) works with programs that use the same zero page locations as the Monitor itself.

Miscellaneous Monitor commands. There are a large number of miscellaneous Monitor commands that I should zip through for you. Some of them, such as I(nverse), N(ormal), val+val, and val-val, have been around since the original Apple II and are still available on the IIgs. Others, such as W(rite) to and R(ead) from cassette tape, have mercifully disappeared on newer machines.

In the same class with I(nverse) and N(ormal) is the control-Y user command. The control-Y command and the use of N as a command separator (much as the colon in used in Applesoft) were discussed at length in the February and March 1985 ***Open-Apples.*** Until the IIgs, the + and - Monitor commands were of little value because only one-byte answers were displayed. The IIgs, on the other hand, can take four-byte operands and display four-byte answers. The IIgs also has a multiply instruction (val*val) that displays eight-byte answers. As before, all of these work with hexadecimal numbers only (where 8+8=10).

While the DOS commands IN# and PR# are the correct way to turn on input devices and printers, even from within the Monitor, the older Monitor commands control-K(eyboard) and control-P(rinter) have been retained. (For some reason, Basic.system commands don't work as well from inside the IIgs Monitor as they do on earlier Monitors.) Likewise, 3D0G (or Q(uit) on the IIgs) usually works better for returning to Applesoft than control-C or control-B, but those commands have been retained as well.

The rest of the miscellaneous commands are new to the IIgs. There are two commands for converting numbers from hex to decimal and back again. To convert to decimal, enter the hex number followed by a equal sign. To convert to hex, enter an equal sign followed by a hex number, like so:

```
*FF=
Decimal-> 255 {+255}
*=255
Hex -> $000000FF
```

Control-T changes the current screen display to text page 1 if you somehow crash into the Monitor while viewing a graphics page. Control-^ (control-shift-6) allows you to change the cursor character. Whatever character you enter after control-^ will become a flashing cursor. Try it. This also works in Applesoft on the IIgs.

=T is a IIgs Monitor command that was mentioned in last month's letters section. It prints the current time on your screen. It has a related command, =T=, which allows you to reset the IIgs clock. I recommend using the control panel instead.

Finally, the IIgs has a "tool locator" command. This command can be used to enter toolbox calls. It begins with a backslash, followed by a number that indicates how may bytes worth of input are needed by the tool, followed by a number that indicates how may bytes of output the tool will return, followed by the input bytes, followed by the two-byte tool number, another backslash, and a U.

Here are two examples of the U command that call the ReadTimeHex and ReadASCIITime tools discussed last month ("Time to look in the toolbox," pages 3.21-22.):

```
*\0 8 D 3\U        (no inputs, 8 bytes of output, tool $0D03)

Tool error-> 0000
0D 00 16 57 0E 03 84 04

*\4 0  0 0 20 0  F 3\U  2000.201F  (4 in, none out, $00002000, tool $0F03)

Tool error-> 0000

00/2000:A0 B4 AF B1 B5 AF B8 B7 A0 B1 B0 BA B0 B3 BA B2- 4/15/87 10:03:2
00/2010:A0 D0 CD 00 00 00 00 00 00 00 00 00 00 00 00 00-6 PM...........
```

This call always returns a "tool error," however, if the error number is zero, no error occurred.

That's the final verse in the new IIgs Monitor. It's becoming a very long song.

Ask (or tell) Uncle DOS

Ladies and gentlemen, start your pencils. On page 3.22 of last month's newsletter, in the middle column, the "AD" after the "00/030A:" should be an "A2."

In regard to last month's letter "All chips not off same block," several subscribers have written in that IIgs memory cards require a type of 256K RAM chip called CAS before RAS. However, apparently not ***all*** *IIgs cards require this kind of chip. If you need chips for a IIgs, I suggest you call or write Microprocessors Unlimited (24000 S Peoria Ave, Beggs, OK 74421 918-267-4961). They sell bulk RAM chips at good prices, provide fast service, send you excellent chip installation instructions, and are keeping track of what chips work in which IIgs cards.*

In my answer to last month's letter "Odd bank out," I asked for help on calling the IIgs memory manager with the Mini-Assembler. I've gotten some good help on that one—you'll read all about it next month. One of the people who responded was Bill Basham of ***Diversi-DOS*** *fame. Basham has already developed three programs for the IIgs that are the most exciting IIgs packages I've seen to date. Not because of stunning graphics or sound, mind you, but because of the way they use the memory manager.* ***Diversi-Cache*** *($35) speeds up Apple 3.5 drives (not UniDisks) by storing the tracks most recently accessed in RAM.* ***Diversi-Key*** *($45) is a memory-resident keyboard macro program that hides itself inside the IIgs and provides macros for* ***all*** *your programs. You have to see this one to believe it. It has lots of bells and whistles, too. With either of the above programs you also get* ***Diversi-Hack****, the wonderful little desk accessory mentioned in this month's lead article that lets you get into the Monitor from the midst of anywhere. These programs work only on the IIgs, of course, and require 512K. (Diversified Software Research, 34880 Bunker Hill, Farmington, MI 48018-2728 313-553-9460).*

Slot 3 RAMdisk rules

Why does *Apple Writer* 2.0 disconnect RAMdisks in slot 3? Can this be changed? My RAMWorks card would be useful as a RAMdisk if *Apple Writer* would stop disconnecting it.

Jerry Hill
FPO Seattle, Wash.

You raise an extremely interesting question. Apple's "ProDOS Technical Note #8" specifies an exact protocol that programs are supposed to use to determine whether a disk device in slot 3 should be disconnected or not. This information is also included in the Addison-Wesley edition of the ***ProDOS Technical Reference Manual*** *(pages 90-91).*

Programs that use both 64K banks of memory, such as ***Apple Writer 2.0****, have to disconnect the ProDOS slot 3 RAMdisk because it also uses the auxiliary 64K bank. If both were active at the same time each would overwrite the other and chaos would reign.*

However, many other RAMdisks, such as the one that came with your auxslot RAM card, either don't use the 64K extended memory area or can be configured not to use it. Since these RAMdisks don't interfere with 128K programs, there is no reason 128K software should disconnect them. Tech Note #8 has specified since late 1984 what RAMdisk developers and what 128K program developers need to do to avoid needless disconnection of slot 3 RAMdisks.

In the beginning, Apple's own software followed the Tech Note #8 protocol—AppleWorks 1.2 follows it exactly. But then something happened. Ken Kashmarek, who recently sent me a ton of information on this issue (much of which I'm using here), thinks that the key event was Apple releasing its own memory card. In order to keep Apple software from working with third-party memory cards, Apple stopped following its own Tech Note, Kashmarek surmises.

According to the Tech Note #8 protocol, 128K software is supposed to look through the ProDOS global-page device list at $BF32-$BF3F and disconnect only those devices that are connected to slot 3, drive 2 and that have the low two bits of their device number set. This translates into units with device numbers of $BF, $BB, $B7, and $B3. The key instructions that accomplish this feat load the device list entries one-by-one, AND each with #$F3, CMP each to #$B3, and branch to disconnect devices that come up "equal."

However, ***Apple Writer 2.0*** *and later, AppleWorks 1.3 and later, and* ***Instant Pascal*** *have two bytes of this protocol changed. All of these products AND with #$70 and CMP with #$30. This has the effect of disconnecting* ***any*** *slot 3 disk device, whether assigned as drive 1 or drive 2, whether RAMdisk, hard disk, or Apple's own disk.*

There are two ways to fix the problem. One is to search through programs that disconnect slot 3 RAMdisks looking for the byte string B9 32 BF 29 70 C9 30. Change the 70 back to F3 and the 30 back to B3. This will make the program follow Apple's published protocol and your RAMdisk will no longer be disconnected.

Another way to avoid the problem is to not assign disk devices to slot 3. Both Applied Engineering and Checkmate Technology have updated their RAMdisk software so that their RAMdisks can appear to be in slot 2.

So, why do ***Apple Writer****, AppleWorks, and* ***Instant Pascal*** *disconnect third-party RAMdisks in slot 3? Does a Fortune 500 company with $800 million in the bank really disregard its own software protocols just to give its RAM card a competitive edge over those from third-party developers? I personally believe the sincerity of Sculley, Yocam, and Gassee when they say that they realize the importance of third-party developers to Apple's success. However, they can't be expected to go over every byte of code in Apple software. And it doesn't take much imagination to picture some low-level product manager being more concerned about achieving sales goals than about Sculley's third-party philosophy.*

Apple's officers need to let the public and its employees know, by actions not words, where they stand. Why doesn't Apple's software follow Apple's own protocols? Lots of people would like to know.

You wanna see a syntax error?

Is there any software that translates what you write in American English to Mexican Spanish? If so, please let me know, I need to constantly get memos, etc., translated.

Don Robinson
Coronado, CA

Dennis replies: The question you ask seems reasonable; translation seems to be no more difficult than looking up words in a dictionary and arranging them into a sentence in the new language. In truth, it is a lot more complex than that; there are verb forms, context, and syntax (among other things) to consider in making the translation. In fact, just translating English into something a computer can understand is difficult enough, to say nothing of then getting the computer to express what it understood in Spanish.

In order to communicate with computers nowadays, we humans have to learn languages such as Applesoft or Pascal, which have very limited vocabularies and very rigid sets of usage rules. In a word, our answer to your question is "no," at least for now, but the issue as to whether such a program will ever be available is interesting.

Translating human language falls into a realm of computer research called artificial intelligence, or "AI." Several excellent books have been written that debate whether computers will ever be able to handle such complicated tasks. One that argues against the ability of a computer to simulate human responses is ***Computer Power and Human Reason****, by Joseph Weizenbaum (W. H. Freeman). Weizenbaum is a pioneer in AI research and was originator of the computer game* ***Eliza****, which provides psychiatrist-like responses to statements entered by a human at the keyboard.*

Another book, which argues against Weizenbaum's dismissal of the practicality of AI research, is ***Gödel, Escher, Bach*** *by Douglas Hofstadter (Vintage Books). Hofstadter leads us through a difficult but enlightening process of attempting to prove that a computer is capable of at least simulating human intelligence. Anyone seriously interested in natural language processing or other AI topics may want to seek out these books, as well as* ***Artificial Intelligence****, by Patrick Henry Winston (Addison-Wesley), and follow their bibliographies to further references.*

The computer language of the AI community is called LISP. A full AI version of LISP requires a lot of computer power. ***LISP****, by Patrick Henry Winston and Berthold Klaus Paul Horn (Addison-Wesley), is a very readable text about the subject. Logo, which at least some of our readers (and more of their children) are familiar with, was derived from LISP.*

Natural language translation is similar in some ways to the command interpretation done by adventure games such as ***Zork****.* ***Zork's*** *interpreter was originally implemented on a Digital Equipment Corp minicomputer using a LISP-like language called MDL, which was then "crunched down" to fit into micros for the commercial versions of the game. For some insight into the design of the interpreter, see "How to Fit a Large Program Into a Small Machine,"* ***Creative Computing****, July 1980, pages 80-87. For additional articles on the design of adventure games that seem to interact with their human players, see*

*the rest of that issue of **Creative Computing** and the December 1980 issue of **Byte**.*

*Readers who are interested in this kind of stuff might like to investigate Polarware's adventure-game interpreter for the Apple II, which is called **Comprehend**. The cost is $95 from Polarware, Box 311, Geneva, IL 60134 800-323-0884.*

CALL -875

I have a program that was written for my old II-Plus and it worked fine—now I have a new IIgs and the program sends me strange machine language screen entries. The program includes both a CALL -875 and a CALL -868 and my guess is that one or the other is causing this. Is this correct? If so, are any of the old calls still valid?

Bob Schmidt
District Heights, Md

CALL -875 (clear current screen line) has not been valid since the introduction of the IIe. Those of you skipping from a II-Plus to a IIgs may have some catching up to do. CALL -875 on the II-Plus jumps into the middle of the screen scroll functions to clear the current line. Apple changed these routines when rewriting the monitor ROM for the IIe. On the IIgs, the code that replaces these routines generates the strange messages you've been seeing.

*CALL -868 (clear to end of line) is still valid. To clear the entire line (entirely from Applesoft) without moving the cursor and **with the 80-column firmware off**, try the sequence:*

```
10 CH = PEEK(36): POKE 36,0: CALL -868: POKE 36, CH
```

*With the 80-column firmware **on**, on the other hand, the correct way to clear the current line is to print a control-Z (CHR$(26)) to the screen (control-L, CHR$(12), will clear the whole screen; control-K, CHR$(11), will clear from the cursor to the end of the screen; and control-], CHR$(29), will clear from the cursor to the end of the current line). That technique is better, for compatibility reasons, than calling a Monitor routine. Too bad it doesn't work with the 80-column firmware off.*

*Apple has published several (slightly different) lists of Monitor addresses that it promises not to change on future Apple models. Any CALL not on one of these lists should be removed from your programs. The version of the list that's my current personal favorite is in the **Apple IIc Technical Reference Manual** (3.5 ROM version) on pages 313-314.*

The great Tinaja Quest

Who or what is "tinaja questing"?

John D. Bishop
Kingston, Ont.

*Don Lancaster often offers a "tinaja quest for two, F.O.B. Thatcher, Arizona" as the grand prize in the contests he devises for his "Ask the Guru" column in **Computer Shopper** ($21/yr, 407 S Washington, Titusville, FL 32796 305-269-3211). I always figured it was something like a snipe hunt, but Dennis called Lancaster's Apple II Hotline number (602-428-4073) and asked. Lancaster said he uses the phrase to get people to call up and ask questions.*

Then he said that "tinaja" is a name for natural basins found in deserts in the Southwest U.S. (usually these basins are private and remote; their name comes from the Spanish word for a large earthen vessel). Like an oasis, a tinaja may be the only source of water and respite in a desert, so "tinaja questing" can be a life and death matter as well as a pleasant recreational pursuit.

*While he had Lancaster on the line, Dennis asked if there was any way to fix the **Apple Writer** "load file to screen" function so that it always used the backslash rather than the current underline character (see November 1986, page 2.77c and February 1987, page 3.2). Lancaster said the complete details were in his **Apple Writer Cookbook** and his May 1987 **Computer Shopper** column (page 244), but the essential details were—bload AWD.SYS at $2000 and then, for version 2.0, 396E:C9 5C EA; for version 2.1, 3974:C9 5C EA. And if you'd like to make **Apple Writer** print through a IIgs serial port, make these patches at the same time—for version 2.0, 4DB0:60, 4F67:10, 4F6E:13; for version 2.1, 4DC7:60, 4F7E:10, 4F85:13 (from Lancaster's March 1987 **Computer Shopper** column, page 108). For the complete how, why, and wherefore of these patches, give Lancaster a jingle.*

APDA erratic on way up

Immediately after reading your September 1986 issue I sent $20 to the Apple Programmers and Developers Association and asked to become a member. In December I called their office and as a result received a receipt for my $20 and a promise that I would soon receive a membership agreement form. In January I called their office and as a result received three membership forms. I promptly signed one and sent it right back. It is now March, and I still don't if I am a member or not.

Joining APDA seems about as difficult as getting technical information out of Apple. Am I doing something wrong? Is anyone else experiencing this same kind of difficulty?

Chuck Zamzow
Battle Creek, Mich.

APDA is alive and well and catching up with itself. You're not the only one who has had a difficult time joining, but APDA assures us you're a member. APDA grew from no members at all in August to 7,000 by the first of the year to 11,000 currently (and still growing). It took awhile for APDA to gear up to the demand. A large backlog developed in December and January because of members signing up at the end of 1986 to take advantage of a free book offer. The backlog was finally cleaned up in February as staff was added. Shipments have been up to speed since then, according to our colleagues at APDA.

Programs, programmers wanted

Softdisk is constantly looking for short Apple II programs of all kinds for publication. In addition, we expect an opening around May 1 for a programmer-writer with extensive Apple technical background and capability of learning C-64 and IBM-PC.

Val J. Golding, Editor-in-Chief
Magazines on Disk
4023 Greenwood Road
Shreveport, LA 71109

*We get several requests a year for information on the "best way" for programmers to get their software published. There are many ways, which range from starting your own software publishing company to donating your work to your local user group's public domain library. In between are a number of outlets that are often overlooked, such as **Softdisk**, **Uptime** (Box 299, Newport, RI 02840), the **Nite Owl Journal** (5734 Lamar, Mission, KS 66202), and the Apple II magazines that print program listings.*

Most people don't realize that the major cost involved with publishing software is marketing. Salespeople, advertising, and dealer discounts are incredibly expensive. If you decide to start your own company you'd better either have deep pockets or an inexpensive marketing scheme.

*One such scheme, which has been very successful with a few software packages, is "shareware." Under this system, you encourage users to make copies of your disk and to distribute them to their friends. However, by means of a screen that appears when the program is started, you ask people who actually use your product to send you a payment for the program. The first product to be successfully distributed this way in the Apple II world was Bill Basham's **Diversi-DOS**. Basham recently told us that about 25 per cent of his **Diversi-DOS** income comes from shareware sales.*

Recently a number of shareware authors joined together in a "programming cooperative" to get more bang from their marketing efforts. The group is called Living Legends Software (1915 Froude Street, San Diego, CA 92107 619/222-3722).

Another route is to sell your software to an established software publishing house and collect royalties on sales. Or maybe you'd like to do contract programming for an established house—Roger Wagner Publishing (P.O. Box 582, Santee, CA 92071), for example, is even now looking for some help from people who own a IIgs and know assembly language.

Pascal RAMdisk loader II

I read with interest the letter (February 1987, page 3.5) from Keith Bernstein regarding a RAMdisk loader for Apple Pascal 1.3. Here's an alternative method to load startup files onto the RAM disk from a 3.5 inch disk.

First, format your RAMdisk using FORMATTER version 1.3 (the Pascal formatter in the ProDOS System Utilities will not work) and transfer all the files you want into the RAM disk manually.

Then use the T)ransfer command saying RAM5:,MYDISK:PASCAL.BACK. The Filer will respond with "Transfer xyz blocks?" Press "N" and specify the number of blocks that your data occupies on the disk; this is found at the bottom of a catalog listing. If you select the default size instead, the full volume, which is mostly empty space, will be transferred. That's rather wasteful.

To use this file, boot up Pascal as usual, then T)ransfer MYDISK:PASCAL.BACK,RAM5:. The Filer will respond with "Remove all files from RAM5:?" After pressing "Y" the RAMdisk will be loaded. You can then press control-reset to boot up from the RAMdisk. Of course, all this can be done from an Exec program such as the one Bernstein wrote about in his letter.

Incidentally, you may be shocked to learn that here in Australia the IIgs retails for around $A3990 and the IIe retrofit for about $A1200.

Daryl Cheshire
Edithvale, Vic

Dennis tried this and says it gives Pascal the same systematic RAMdisk startup/shutdown procedures we described for other operating systems in December 1986 ("RAM Van Lines," page 2.87). To conserve space, Dennis suggests the RAMdisk should be K)runched with the Filer before determining it's block size and saving out it's contents. And, as you point out, this technique wipes out any files already on the RAMdisk when a "restore" is done.

The prices you quote are about $2850 and $850 in U.S. dollars. If your import taxes on goods from Singapore (where the IIgs is manufactured) aren't

any higher than those in the U.S., the difference must be freight. (Seriously, Apple is doing a magnificent job—for a U.S. company—in developing products for a world-wide market. Are you sure at least part of the difference isn't Australian taxes? There's no other legitimate reason for that large a price difference—it just encourages black market transactions.)

Stop double RAM load

Alan Bird's "Don't pass go" program for AppleWorks (November 1986, page 2.75 and December 1986, page 2.84) has been most helpful, but I have run into a bit of a snag with AppleWorks 2.0. This version automatically loads itself into RAM at startup. Since I usually already have the program on my RAMdisk, there's no reason to load it again. Do you know how to keep it from doing that without having to press the escape key?

Harlan R. Davis
Bolingbrook, Ill.

Try this:

```
10  REM *** Don't Pass Go for RAMdisk ***

20  TEXT : HOME : VTAB 10
25  D$=CHR$(4) : F$="APLWORKS.SYSTEM"
30  PRINT D$;"BLOAD";F$;",A$2000,TSYS"
40  IF PEEK(8250) < > 57 THEN PRINT
      "Program requires AppleWorks 2.0." : END
45  PRINT "Patching AppleWorks"

50  POKE 14468,44  : REM no space bar

60  POKE 14148,208
61  POKE 14149,19  : REM no return for date

70  FOR ADR=13271 TO 13273
71  : POKE ADR,234
72  NEXT           : REM no RAMdisk preload

100 PRINT D$;"BSAVE";F$;",A$2000,TSYS"
```


Open-Apple

is written, edited, published, and

© Copyright 1987 by Tom Weishaar

Business Consultant	**Richard Barger**
Technical Consultant	**Dennis Doms**
Circulation Manager	**Sally Tally**
Business Manager	**Sally Dwyer**

Most rights reserved. All programs published in ***Open-Apple*** are public domain and may be copied and distributed without charge. Apple user groups and significant others may reprint articles from time to time by specific written request. Requests and other editorial material, including letters to Uncle DOS, should be sent to:

Open-Apple
P.O. Box 7651
Overland Park, Kansas 66207 U.S.A.

Published monthly since January 1985. World-wide prices (in U.S. dollars; airmail delivery included at no additional charge): $24 for 1 year; $44 for 2 years; $60 for 3 years. All single back issues are currently available for $2 each; bound, indexed editions of Volume 1 and Volume 2 are $14.95 each. Volumes end with the January issue; an index for the prior volume is included with the February issue. Please send all subscription-related correspondence to:

Open-Apple
P.O. Box 6331
Syracuse, N.Y. 13217 U.S.A.

Subscribers in Australia and New Zealand should send subscription correspondence to ***Open-Apple***, c/o Cybernetic Research Ltd, 576 Malvern Road, Prahran, VIC 3181, AUSTRALIA.

Open-Apple is available on disk from Speech Enterprises, P.O. Box 7986, Houston, Texas 77270 (713-461-1666).

Unlike most commercial software, ***Open-Apple*** is sold in an unprotected format for your convenience. You are encouraged to make back-up archival copies or easy-to-read enlarged copies for your own use without charge. You may also copy ***Open-Apple*** for distribution to others. The distribution fee is 15 cents per page per copy distributed.

WARRANTY AND LIMITATION OF LIABILITY. I warrant that most of the information in ***Open-Apple*** is useful and correct, although drivel and mistakes are included from time to time, usually unintentionally. Unsatisfied subscribers may return issues within 180 days of delivery for a full refund. Please include a note from your parents or children confirming that all archival copies have been destroyed. The unfullfilled portion of any paid subscription will be refunded on request. MY LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for any damages in excess of the fees paid by a subscriber.

ISSN 0885-4017 **Source Mail: TCF238**
Printed in the U.S.A. **CompuServe: 70120,202**

Blink and it's gone

I use a C-Vue LCD screen a lot. It's like eating tofu—you eventually develop a taste for low contrast. But the "insert" cursor in AppleWorks (the blinking underline) is impossible to find. The "replace" cursor never gets lost. Most of the time I use insert mode. Is it possible to patch AppleWorks so that the insert mode cursor is a blinking inverse block?

Tom Meyer
Highlands, NC

To replace the underline character with a new value, use:

```
BLOAD APLWORKS.SYSTEM,A$2000,TSYS
CALL -151
<addr>:<val>
3D0G
BSAVE APLWORKS.SYSTEM,A$2000,TSYS
```

where <addr> is 2D80 for version 1.2, 2D8B for version 1.3, or 2DA1 for version 2.0, and <val> is the ASCII value you want for your cursor (DF gives the underline). We tested this patch using "FF" for the value, which gives a flashing checkerboard box for the insert cursor, and using "20," which gives an inverse box. I think you'll be able to see 20 better, but you may have trouble distinguishing it from the replace cursor—the only difference will be that the insert cursor blinks faster. The value for the replace cursor itself is generated by code that also appears to initialize other routines; we decided not to mess with it.

AppleWorks reset

Please give us a patch for AppleWorks that makes reset work when AppleWorks hangs. I'm looking for a more general solution than those you mentioned at the end of your answer to "Insert system disk and..." in April (page 3.18)—something that would work without *MacroWorks* or with *PinPoint*, etc. There must be a warm start address in AppleWorks somewhere!

Thom Ryan
Toronto, Ont.

*Back in June 1986, page 2.33, we published "An AppleWorks Rescue Routine" that works with all AppleWorks versions prior to 2.0. It has the advantage of being useful **after** AppleWorks hangs. The other techniques we mentioned in June involve your doing something special **before** AppleWorks hangs so that you'll be able to recover. Here's an instant replay of the June 1986 routine, along with some new information on how to make it work with AppleWorks 2.0:*

```
Press control-reset to get into the Monitor.
  (If you can't get to the Monitor, go to jail.)
*C073:0
*3 control-P return
  (If this doesn't get you 80-columns, go to jail.)
*2F0:2C 83 C0 2C 83 C0 4C
*:33 10   <--for AppleWorks 1.1 through 1.3
*:27 11   <--for AppleWorks 2.0
*2F0G
```

If the main menu appears messed up, just press escape. Save any files you have on the desktop and reboot after using this technique.

More mail merge categories

There is a peculiarity to the mail merge function of Appleworks 2.0 people should know about. I couldn't understand why some of the categories from my database were not being picked up and inserted, as specified, in my form letter. I then began to wonder about the fact that the mail merge data must be printed to the clipboard using a tables-style report. That report format has a default platen width of 8 inches. Sure enough, changing the platen width to something greater (e.g., 17 inches) allowed all of my categories to be printed to the clipboard. Changing the characters per inch to 17 can accomplish the same thing.

I hope that this will save someone a headache or two.

William J. Linville
Terre Haute, Ind.

Telephone feedback

I want to thank Jim Hercules for his *inspired* AppleWorks phone dialer (April 1987, page 3.18). Because of his discovery that you can set up a modem as a printer in AppleWorks, I now have an incredibly fast and easy to use phone dialer. But mine's in a spreadsheet file. Row 1 is a name, Row 2 is that name's phone number, including 1 and area code if needed; Row 3 is the next name, Row 4 that person's phone number, and so on. I start the phone number with a " to make it a label so I can include commas for pauses and dashes for readability. I can open-apple-F(ind) the name I want to dial, use the down arrow key to highlight the number below the name, then print that row to the modem. I can scroll back and forth through my phone list, insert, delete, or make changes. Hercules' suggestion is one of those outstandingly useful tips that I've come to expect from ***Open-Apple*** and its readers.

Thanks also to Tony Bond (page 3.20) for the suggestion to copy an AppleWorks 1.3 SEG.PR file onto an AppleWorks 2.0 disk in order to get control-@ entered as a printer command. This seems to work fine, but if you're using Applied Engineering's Apple-Works 2 Expander, you'll have to reinstall it (it uses the SEG.PR file for information of its own). That means reinstalling *Super MacroWorks*, too.

I have a suggestion for Jim Thornburg's problems with using date categories in his AppleWorks genealogy data base. AppleWorks' two-digit year just doesn't make it in genealogy, but dates can be effectively manipulated manually for some very useful reports. Enter birth dates in three categories: Birth-Year (1860), Birth-Month (02), and Birth-Day (29). Do the same with death dates. A chronological sort of all records by birth date is easily accomplished using multiple sorts—sort days first, then months, then years. You can also print reports with calculated categories that subtract Birth-Year from Death-Year and Birth-Month from Death-Month to give approximate age at death.

Another option is to enter dates in a single category in the following format: 1875-07-22. This format sorts nicely *alphabetically* in forward or reverse order. So far I've found no need for AppleWorks' automatic date feature in my genealogy data base, despite its usefulness in other situations.

C. L. Roberts
Lafayette, Calif.